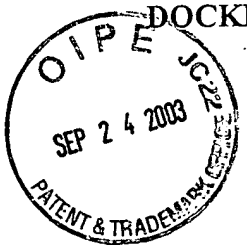
DOCKET NO. US 000206 (PHIL06-00206) PATENT



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Krishnamurthy Vaidyanathan, et al.
Serial No.:	09/639,149
Filed:	August 16, 2000
For:	REPROGRAMMABLE APPARATUS SUPPORTING THE PROCESSING OF A DIGITAL SIGNAL STREAM AND METHOD
Group No.:	2181
Examiner:	Raymond N. Phan

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

APPEAL BRIEF

The Appellants have appealed to the Board of Patent Appeals and Interferences from the decision of the Examiner dated April 24, 2003, finally rejecting Claims 1-17. The Appellants filed a Notice of Appeal on July 21, 2003. The Appellants respectfully submit this brief on appeal, in triplicate, with a statutory fee of $320.00.

REAL PARTY IN INTEREST

This application is currently owned by Philips Electronics North America Corporation as indicated by an assignment recorded on August 16, 2000 in the Assignment Records of the United States Patent and Trademark Office at Reel 011176, Frame 0066.

RELATED APPEALS AND INTERFERENCES

There are no known appeals or interferences that will directly affect or be directly affected by or have a bearing on the Board's decision in this pending appeal.

STATUS OF CLAIMS

Claims 1-17 stand rejected pursuant to a final Office Action mailed April 24, 2003. Claims 1-17 are presented for appeal. Claims 1-17 are shown in Appendix A.

STATUS OF AMENDMENTS

The Appellants submitted an Amendment and Response to Office Action on June 6, 2003. The Examiner refused to enter the amendment, asserting that it failed to place the application in better form for appeal by materially reducing or simplifying the issues for appeal.

SUMMARY OF INVENTION

According to one embodiment, a device such as a channel decoder includes a digital signal processor (element 310), a programmable bridge (element 350), and a set of functional units (elements 320). (*Application, Page 5, Lines 11-17*). The functional units represent any of a

wide variety of functions, such as a baseband modem, a tuner, an error corrector, or a filter. (*Application, Page 6, Lines 1-3*).

The programmable bridge supports the communication of information between the digital signal processor and/or the functional units. (*Application, Page 5, Lines 11-21*). In an example embodiment, the programmable bridge includes interface registers (elements 440 and 450) and register transfer units (element 420). (*Application, Page 5, Lines 28-29; Page 6, Lines 16-18*). The interface registers communicate information to and receive information from the functional units and the digital signal processor. (*Application, Page 6, Lines 5-11*). The register transfer units support the exchange of information between the interface registers. (*Application, Page 6, Lines 16-18*).

The programmable bridge also allows the information to be transformed or altered as it is being transferred between the digital signal processor and/or the functional units. In the example embodiment, the programmable bridge also includes reconfigurable datapath units (element 430). (*Application, Page 7, Lines 15-16*). The reconfigurable datapath units transform information as it is transferred between the interface registers. (*Application, Page 7, Lines 16-17*).

In addition, the programmable bridge can be reprogrammed to change the way in which information is routed and/or the way in which the information is transformed. (*Application, Page 5, Lines 22-26*). In this way, the operation of the channel decoder or other device can be altered by changing the programmable bridge. (*Application, Page 8, Lines 10-13*).

STATEMENT OF ISSUES

Are Claims 1-17 unpatentable under 35 U.S.C. §§ 102(e), 103?

GROUPING OF CLAIMS

Pursuant to 37 C.F.R. § 1.192(c)(7), the Appellants request that the following claims be grouped together for purposes of this appeal:

(1) Group 1: Claims 1, 2, 5-7, 10-12, and 14-17.

Claims 1, 2, 5-7, 10-12, and 14-17 may be deemed to stand or fall together for purposes of this appeal.

(2) Group 2: Claims 3, 4, 8, 9, and 13.

Claims 3, 4, 8, 9, and 13 may be deemed to stand or fall together for purposes of this appeal.

The Appellants submit that the explanations provided in the Argument section do not merely point out differences between the claims but present arguments as to the separate patentability of each claim as required by 37 C.F.R. § 1.192(c)(7) and M.P.E.P. § 1206.

ARGUMENT

The rejections of Claims 1-17 under 35 U.S.C. § 102(e) and § 103(a) are improper and should be withdrawn.

A. OVERVIEW

Claims 1-4, 11-13, and 17 stand rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,233,637 to Smyers et al. ("*Smyers*"). Claims 5-10 and 14-16 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Smyers* in view of "Applicant Admitted Prior Art" ("*AAPA*"). A copy of *Smyers* is provided in Appendix B.

B. STANDARD

1. 35 U.S.C. § 102

A cited prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. (*MPEP § 2131; In re Bond, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990)*). Anticipation is only shown where each and every limitation of the claimed invention is found in a single cited prior art reference. (*MPEP § 2131; In re Donohue, 766 F.2d 531, 534, 226 U.S.P.Q. 619, 621 (Fed. Cir. 1985)*).

2. 35 U.S.C. § 103

In *ex parte* examination of patent applications, the Patent Office bears the burden of

establishing a *prima facie* case of obviousness. (*MPEP § 2142; In re Fritch, 972 F.2d 1260, 1262, 23 U.S.P.Q.2d 1780, 1783 (Fed. Cir. 1992)*). The initial burden of establishing a *prima facie* basis to deny patentability of a claimed invention is always upon the Patent Office. (*MPEP § 2142; In re Oetiker, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); In re Piasecki, 745 F.2d 1468, 1472, 223 U.S.P.Q. 785, 788 (Fed. Cir. 1984)*). Only when a *prima facie* case of obviousness is established does the burden shift to the applicant to produce evidence of nonobviousness. (*MPEP § 2142; In re Oetiker, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); In re Rijckaert, 9 F.3d 1531, 1532, 28 U.S.P.Q.2d 1955, 1956 (Fed. Cir. 1993)*). If the Patent Office does not produce a *prima facie* case of unpatentability, then without more the applicant is entitled to grant of a patent. (*In re Oetiker, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); In re Grabiak, 769 F.2d 729, 733, 226 U.S.P.Q. 870, 873 (Fed. Cir. 1985)*).

A *prima facie* case of obviousness is established when the teachings of the prior art itself suggest the claimed subject matter to a person of ordinary skill in the art. (*In re Bell, 991 F.2d 781, 783, 26 U.S.P.Q.2d 1529, 1531 (Fed. Cir. 1993)*). To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed invention and the reasonable expectation of success must both be found in the

prior art, and not based on applicant's disclosure. (*MPEP § 2142*).

C. THE *SMYERS* REFERENCE

Smyers recites a system for providing a bi-directional path between an application and a data bus. (*Abstract*). The system includes an isochronous data pipe (element 20) and an asynchronous data pipe (element 26), and each data pipe includes a register set. (*Col. 3, Lines 30-43*). The register sets (elements 24 and 28) are programmed by the application. (*Col. 4, Lines 47-48*). The register sets contain information controlling how a data transfer should occur in the asynchronous data pipe and defining how data is to be processed in the isochronous data pipe. (*Col. 4, Lines 47-50; Col. 11, Lines 49-53*). The system also includes a set of control registers (element 38). (*Col. 3, Lines 38-43*).

D. GROUP 1 (CLAIMS 1, 2, 5-7, 10-12, AND 14-17)

1. 35 U.S.C. § 102 REJECTION

Claim 1 recites:

> A bridge comprising:
>
> a plurality of interface registers that are configured to facilitate communication of data with a plurality of function units, and
>
> a plurality of register transfer units, operably coupled to the plurality of interface registers, that facilitate transfers of data among interface registers of the plurality of interface registers.

Smyers fails to anticipate all elements of Claim 1. In particular, *Smyers* fails to anticipate "register transfer units" that "facilitate transfers of data among interface registers" as recited in

Claim 1.

First, the Examiner asserts that *Smyers* recites these elements of Claim 1 at column 3, line 64 through column 4, line 7. (*04/24/03 Office Action, Page 3, Second paragraph*). This portion of *Smyers* recites using First In, First Out (FIFO) queues to transport information between the data pipes (elements 20 and 26) and a link core (element 44).

This portion of *Smyers* contains no recitation that the FIFO queues are used to transport information between registers. In particular, the FIFO queues do not transport data to or from the control registers (element 38). Also, the register sets (elements 24 and 28) contain information controlling how a data transfer should occur in the asynchronous data pipe and defining how data should be processed in the isochronous data pipe. *Smyers* contains no recitation that the FIFO queues are used to transport the contents of one register set to the other register set. In addition, *Smyers* contains no recitation that the FIFO queues are used to transport data between registers in one of the register sets.

Because the FIFO queues do not transport data between registers, the FIFO queues in *Smyers* cannot anticipate a "plurality of register transfer units" that "facilitate transfers of data among interface registers" as recited in Claim 1.

Second, the Examiner asserts that *Smyers* discloses "transferring data between the registers in the data pipes" at column 12, line 28 through column 13, line 65. (*04/24/03 Office Action, Page 6, Paragraph 8*). However, the issue is not whether *Smyers* simply discloses transferring data between registers. To establish anticipation, the Examiner must show *Smyers* discloses a "plurality of register transfer units" that facilitate transfers of data among "interface

registers" as recited in Claim 1. The Examiner has not made this showing.

The second portion of *Smyers* cited by the Examiner describes the register set in the isochronous data pipe. (*Col. 11, Lines 49-53*). In particular, this portion of *Smyers* refers to three different registers: a "current stack register" (*Col. 12, Lines 28-37*), a "source register" (*Col. 12, Lines 54-56*), and a "destination register" (*Col. 12, Lines 56-58*). As described above, the Examiner asserts that the FIFO queues in *Smyers* anticipate the "plurality of register transfer units" recited in Claim 1. The Examiner's reference to this portion of *Smyers* either means that the Examiner (1) is now relying on the registers in the isochronous data pipe to anticipate the "register transfer units" of Claim 1, or (2) is asserting that the FIFO queues transport information between the registers in the isochronous data pipe.

If the Examiner is relying on the specific registers in the isochronous data pipe to anticipate the "register transfer units" of Claim 1, this is inconsistent with the Examiner's previous assertion that the FIFO queues anticipate this element of Claim 1. If the Examiner is asserting that the FIFO queues transport information between the registers in the isochronous data pipe, this is an improper interpretation of *Smyers*. *Smyers* contains absolutely no mention of a FIFO queue transporting data between any of the described registers in the isochronous data pipe. As a result, in either case, the Examiner cannot show that *Smyers* anticipates the use of "register transfer units" as recited in Claim 1.

For these reasons, *Smyers* fails to anticipate the Applicants' invention recited in Claim 1 (and Claims 2-5 depending from Claim 1). For similar reasons, *Smyers* fails to anticipate the Applicants' invention recited in Claim 11 (and Claims 12-17 depending from Claim 11).

2. 35 U.S.C. § 103 RJECTION

Claim 5 depends from Claim 1, and Claims 14-16 depend from Claim 11. Because *Smyers* fails to disclose, teach, or suggest the Applicants' invention recited in Claims 1 and 11, Claims 1 and 11 are patentable. As a result, Claims 5 and 14-16 are allowable due to their dependence from allowable base claims.

Regarding Claim 6, *Smyers* fails to disclose, teach, or suggest a "plurality of register transfer units" that facilitate "transfers of data among interface registers" as recited in Claim 6. The Examiner only relies on the *AAPA* to the extent that it recites a channel decoder. The Examiner does not rely on the *AAPA* as reciting a "plurality of register transfer units" that facilitate "transfers of data among interface registers."

For these reasons, the proposed *Smyers-AAPA* combination fails to disclose, teach, or suggest the Applicants' invention recited in Claim 6 (and Claims 7-10 depending from Claim 6).

E. GROUP 2 (CLAIMS 3, 4, 8, 9, AND 13)

Claim 3 recites:

> The bridge of claim 1, further comprising:
> at least one datapath unit, operably coupled to the plurality of register transfer units, that facilitates a transformation of at least one data item of the data that is transferred among the interface registers.

Smyers fails to anticipate all elements of Claim 3. In particular, *Smyers* fails to anticipate a datapath unit that "facilitates a transformation of at least one data item of the data that is transferred among the interface registers" as recited in Claim 3.

The Examiner asserts that *Smyers* recites these elements at column 6, lines 16-39. (*04/24/03 Office Action, Page 3, Third paragraph*). This portion of *Smyers* recites how FIFO queues (elements 30, 32, 34), a multiplexer (element 40), and a demultiplexer (element 42) are used to facilitate communication between the data pipes (elements 24, 28) and the link core (element 44). This portion of *Smyers* lacks any mention of facilitating a "transformation" of a "data item" that is "transferred among the interface registers." In fact, this portion of *Smyers* contains absolutely no mention of transforming data in any way. It also completely fails to mention transferring data among registers.

For these reasons, *Smyers* fails to anticipate the Applicants' invention recited in Claim 3 (and Claim 4 depending from Claim 3). For similar reasons, *Smyers* fails to anticipate the Applicants' invention recited in Claims 8 and 13 (and Claim 9 depending from Claim 8). In addition, Claims 3, 4, 8, 9, and 13 are patentable due to their dependence from allowable base claims.

F. SUMMARY

For the reasons given above, *Smyers* fails to anticipate the Applicants' invention recited in Claims 1-4, 11-13, and 17. The proposed *Smyers-AAPA* combination also fails to disclose, teach, or suggest the Applicants' invention recited in Claims 5-10 and 14-16. As a result, Claims 1-17 are patentable over *Smyers* and the *Smyers-AAPA* combination.

CONCLUSION

The Appellants have demonstrated that the present invention as claimed is clearly distinguishable over the prior art cited of record. Therefore, the Appellants respectfully request the Board of Patent Appeals and Interferences to reverse the final rejection of the Examiner and instruct the Examiner to issue a notice of allowance of all claims.

The Appellants have enclosed a check in the amount of $320.00 to cover the cost of this Appeal Brief. The Appellants do not believe that any additional fees are due. However, the Commissioner is hereby authorized to charge any additional fees or credit any overpayments to Davis Munck Deposit Account No. 50-0208. No extension of time is believed to be necessary. If an extension of time is needed, however, the extension is requested. Please charge the fee for the extension to Deposit Account No. 50-0208.

Respectfully submitted,

DAVIS MUNCK, P.C.

Date: Sept. 22, 2003

William A. Munck
Registration No. 39,308

P.O. Drawer 800889
Dallas, Texas 75380
(972) 628-3600 (main number)
(972) 628-3616 (fax)
E-mail: *wmunck@davismunck.com*

APPENDIX A

PENDING CLAIMS

1. A bridge comprising:

a plurality of interface registers that are configured to facilitate communication of data with a plurality of function units, and

a plurality of register transfer units, operably coupled to the plurality of interface registers, that facilitate transfers of data among interface registers of the plurality of interface registers.

2. The bridge of claim 1, further comprising:

an instruction memory that is configured to contain register transfer instructions, and

wherein the operable coupling of the plurality of register transfer units and the plurality of function units is effected via the register transfer instructions.

3. The bridge of claim 1, further comprising:

at least one datapath unit, operably coupled to the plurality of register transfer units, that facilitates a transformation of at least one data item of the data that is transferred among the interface registers.

4. The bridge of claim 3, further comprising:

an instruction memory that is configured to contain register transfer instructions, and

wherein the operable coupling of the plurality of register transfer units and the plurality of function units and the at least one datapath unit is effected via the register transfer instructions.

5. The bridge of claim 1, wherein:

at least one of the function units is a programmable digital signal processor.

6. A signal processing system comprising:

a receiver that is configured to provide a digital input stream,

a channel decoder, operably coupled to the receiver, that is configured to decode the digital input stream into a decoded signal stream, and

a user application, operably coupled to the channel decoder, that is configured to render an output corresponding to a channel of the digital input stream based on the decoded signal stream,

wherein the channel decoder comprises a bridge comprising:

a plurality of interface registers, each associated with a processing unit of a plurality of processing units, and

a plurality of register transfer units, operably coupled to the plurality of interface registers, that facilitate:

transfers of data among interface registers of the plurality of interface registers,

transfers of data of the digital input stream among interface registers of the plurality of interface registers, and

transfers of data from the interface registers to provide the decoded signal stream.

7. The signal processing system of claim 6, wherein the channel decoder further comprises:

an instruction memory that is configured to contain register transfer instructions, and

wherein the operable coupling of the plurality of register transfer units and the plurality of processing units is effected via the register transfer instructions.

8. The signal processing system of claim 6, further comprising:

at least one datapath unit, operably coupled to the plurality of register transfer units, that facilitates a transformation of at least one data item of the data that is transferred among the interface registers.

9. The signal processing system of claim 8, wherein the channel decoder further comprises:

an instruction memory that is configured to contain register transfer instructions, and

wherein the operable coupling of the plurality of register transfer units and the plurality of processing units and the at least one datapath unit is effected via the register transfer instructions.

10. The signal processing system of claim 6, wherein:

at least one of the processing units is a programmable digital signal processor.

11. A method, comprising:

receiving data at a plurality of interface registers, the interface registers operable to communicate with a plurality of function units;

allowing at least one of the function units to process the data in at least one of the interface registers;

communicating the data among at least two of the interface registers using a plurality of register transfer units; and

allowing at least one of the function units to further process the data in at least one of the interface registers.

12. The method of Claim 11, wherein communicating the data among at least two of the interface registers comprises using register transfer instructions in an instruction memory to communicate the data among at least two of the interface registers.

13. The method of Claim 11, further comprising transforming at least one data item of the data that is transferred among the interface registers.

14. The method of Claim 11, wherein at least one of the function units is a programmable digital signal processor.

15. The method of Claim 11, wherein receiving data comprises receiving a digital input stream from a receiver.

16. The method of Claim 15, further comprising generating a decoded signal stream using the digital input stream.

17. The method of Claim 16, further comprising communicating the decoded signal stream to a user application operable to render an output corresponding to a channel of the digital input stream based on the decoded signal stream.

APPENDIX B

Smyers Reference

U.S. Patent No. 6,233,637

US006233637B1

(12) **United States Patent**
Smyers et al.

(10) **Patent No.: US 6,233,637 B1**
(45) **Date of Patent: *May 15, 2001**

(54) **ISOCHRONOUS DATA PIPE FOR MANAGING AND MANIPULATING A HIGH-SPEED STREAM OF ISOCHRONOUS DATA FLOWING BETWEEN AN APPLICATION AND A BUS STRUCTURE**

(75) Inventors: **Scott D. Smyers**, Los Gatos; **Bruce Fairman**, Woodside, both of CA (US); **Hisato Shima**, Tokyo (JP)

(73) Assignees: **Sony Corporation**, Tokyo (JP); **Sony Electronics, Inc.**, Park Ridge, NJ (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/612,322**

(22) Filed: **Mar. 7, 1996**

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **710/129**; 710/128
(58) **Field of Search** 395/306, 307, 395/309, 275, 800, 200.12, 200.14, 200.19, 200.2, 284, 285, 286; 370/85.15, 60.1, 58.1, 85.3, 85.12, 85.13, 94.2, 94.3

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,836,722	9/1974	Muller et al.	179/15 BS
3,906,484	9/1975	Melvin, Jr. et al.	340/347 DD
4,218,756	8/1980	Fraser	364/900
4,379,294	4/1983	Sutherland et al.	340/825.5
4,395,710	7/1983	Einolf, Jr. et al.	340/825.5
4,409,656	10/1983	Anderson et al.	364/200
4,493,021	1/1985	Agrawal et al.	364/200
4,633,392	12/1986	Vincent et al.	364/200
4,641,307	2/1987	Russell	370/60
4,739,323	4/1988	Miesterfeld et al.	340/825.5
4,897,783	1/1990	Nay	364/200

(List continued on next page.)

OTHER PUBLICATIONS

"The Parallel Protocol Engine" Matthias Kaiserswerth, IEEE/ACM Transactions on Networking, Dec. 1993, New York, pp. 650–663.

"The Programmable Protocol VLSI Engine (PROVE)" A.S. Krishnakumar, W.C. Fischer, and Krishan Sabnani, IEEE Transactions on Communications, Aug. 1994, New York, pp. 2630–2642.

"A Bus on a Diet–The Serial Bus Alternative" Michael Teener, ComCon92, Feb. 24–28, 1992, pp. 316–321.

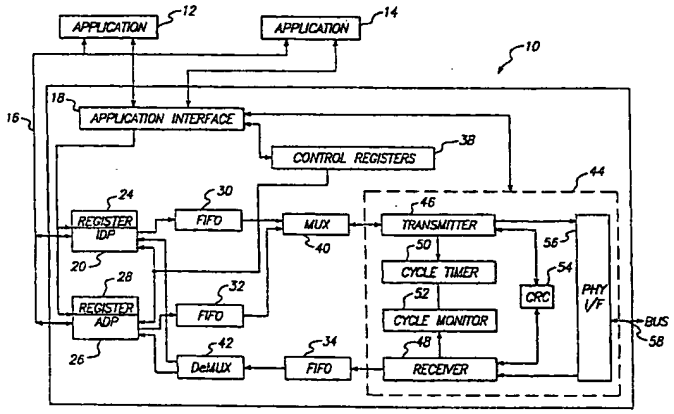
(List continued on next page.)

Primary Examiner—Robert Beausoleil
Assistant Examiner—Raymond N Phan
(74) *Attorney, Agent, or Firm*—Haverstock & Owens LLP

(57) **ABSTRACT**

An isochronous data pipe provides a bi-directional path for data between an application and a bus structure. The isochronous data pipe includes the ability to send, receive and perform manipulations on any isochronous stream of data, including data on any number of isochronous channels. The isochronous data pipe is a programmable sequencer that operates on the stream of isochronous data as it passes through the isochronous data pipe. The isochronous data pipe is programmed by an application to perform specific operations on the stream of data before the data is either transmitted across the bus structure or sent to the application, thereby pre-processing and manipulating the data before it is delivered to its destination. The operations are performed on both the packet header and the data field of the data packet. The isochronous data pipe can be stopped and started on the occurrence of specific events. In an alternate embodiment of the present invention, the isochronous data pipe is programmed to send and receive both isochronous and asynchronous data, including generating requests and appropriate packet headers.

24 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

4,972,470		11/1990	Farago 380/3
4,998,245		3/1991	Tanaka et al. 370/85.1
5,008,879		4/1991	Fischer et al. 370/85.2
5,117,070		5/1992	Ueno et al. ..... 178/2 R
5,191,418		3/1993	Tran 358/142
5,276,684		1/1994	Pearson 370/94.1
5,325,510		6/1994	Frazier 395/425
5,343,469		8/1994	Ohshima 370/85.1
5,359,713		10/1994	Moran et al. 395/200
5,361,261		11/1994	Edem et al. 370/85.3
5,400,340		3/1995	Hillman et al. 370/105.3
5,402,419		3/1995	Osakabe et al. 370/85.1
5,412,698		5/1995	Van Brunt et al. 375/373
5,420,573		5/1995	Tanaka et al. 340/825.24
5,444,709		8/1995	Riddle 370/94.1
5,465,402		11/1995	Ono et al. 455/161.2
5,487,153		1/1996	Hammerstrom et al. 395/250
5,493,570		2/1996	Hillman et al. 370/105.3
5,497,466		3/1996	Roden et al. 395/306
5,499,344		3/1996	Elnashar et al. 395/250
5,506,846		4/1996	Edem et al. 370/94.2
5,509,126	*	4/1996	Oprescu et al. 395/307
5,519,701		5/1996	Colmant et al. 370/60.1
5,524,213		6/1996	Dais et al. 395/200.17
5,526,353		6/1996	Henley et al. 370/60.1
5,533,018		7/1996	DeJager et al. 370/60.1
5,535,208	*	7/1996	Kawakami et al. 370/84
5,537,408		7/1996	Branstad et al. 370/79
5,544,324	*	8/1996	Edem et al. 395/200.17
5,546,389		8/1996	Wippendbeck et al. 370/60
5,546,553		8/1996	Robertson et al. 395/405
5,548,587		8/1996	Bailey et al. 370/60.1
5,550,802	*	8/1996	Worsley et al. 370/13
5,559,796		9/1996	Edem et al. 370/60
5,559,967		9/1996	Oprescu et al. 395/285
5,566,174		10/1996	Sato et al. 370/84
5,586,264		12/1996	Belknap et al. 395/200.08
5,594,732		1/1997	Bell et al. 370/401
5,594,734		1/1997	Worsley et al. 370/395
5,602,853		2/1997	Ben-Michael et al. 370/474
5,603,058		2/1997	Belknap et al. 395/855
5,615,382	*	3/1997	Gavin et al. 395/800
5,617,419		4/1997	Christensen et al. 370/471
5,619,646		4/1997	Hoch et al. 395/200.01
5,632,016		5/1997	Hoch et al. 395/200.02
5,640,392	*	6/1997	Hayashi 370/395
5,640,592		6/1997	Rao 710/5
5,646,941		7/1997	Nishimura et al. 370/389
5,647,057	*	7/1997	Roden et al. 395/275
5,652,584		7/1997	Yoon 341/89
5,655,138		8/1997	Kikinis 395/808
5,664,124	*	9/1997	Katz et al. 395/309
5,668,948		9/1997	Belknap et al. 395/200.61
5,684,954		11/1997	Kaiserswerth et al. 395/200.2
5,687,174	*	11/1997	Edem et al. 370/446
5,687,316		11/1997	Graziano et al. 395/200.2
5,689,244		11/1997	Iijima et al. 340/825.07
5,692,211	*	11/1997	Gulick et al. 395/800.01
5,694,555		12/1997	Morriss et al. 395/280
5,696,924		12/1997	Robertson et al. 395/412
5,706,439		1/1998	Parker 395/200.17
5,708,779		1/1998	Graziano et al. 395/200.8
5,710,773		1/1998	Shiga 370/512
5,752,076		5/1998	Munson 395/825
5,758,075		5/1998	Graziano et al. 395/200.8
5,761,430	*	6/1998	Gross et al. 709/250
5,761,457		6/1998	Gulick 395/308
5,774,683		6/1998	Gulick 395/309
5,781,599	*	7/1998	Shiga 375/376
5,787,256		7/1998	Marik et al. 395/200.68
5,787,298		7/1998	Broedner et al. 395/750.05
5,790,236		10/1999	Galloway et al. 395/500.44
5,793,953		8/1998	Yeung et al. 395/200.8
5,799,041		8/1998	Szkopek et al. 375/259
5,812,883		9/1998	Rao 395/8.94
5,815,678		9/1998	Hoffman et al. 395/309
5,828,093		10/1998	Sethuram et al. 395/817
5,828,416		10/1998	Ryan 348/512
5,832,245		11/1998	Gulick 395/309
5,835,726		11/1998	Shwed et al. 709/229
5,835,793		11/1998	Li et al. 395/898
5,848,253		12/1998	Walsh et al. 395/309
5,872,983		2/1999	Walsh et al. 395/750.01
5,875,312		2/1999	Walsh et al. 395/309
5,884,103		3/1999	Terho et al. 710/72
5,887,145		3/1999	Harari et al. 395/282
5,938,752		8/1999	Leung et al. 710/126
5,946,298		8/1999	Okuyama 370/232
5,987,126	*	11/1999	Okuyama et al. 380/5
5,991,520		11/1999	Smyers et al. 395/280
6,085,270	*	7/2000	Gulick 710/100

OTHER PUBLICATIONS

"Local Area Network Protocol for Autonomous Control of Attached Devices" Software Patent Institute, 1995, 1996.

"Architecture for High Performance Transparent Bridges" Software Patent Institute, 1995, 1996.

"Access to High–Speed LAN via Wireless Media" Software Patent Institute, 1995, 1996.

"Asynchronous Transfer Mode" Julia L. Heeter, Dec. 12, 1995.

"The SerialSoft IEEE 1394 Developer Tool" Skipstone.

"Data link driver program design for the IBM token ring network PC adapter" Gee–Swee Poo and Wilson Ang, Computer Communications, 1989, London, Great Britain, pp. 266–272.

"Fiber Channel (FCS)/ATM interworking: A design solution" A. Anzaloni, M. De Sanctis, F. Avaltroni, G. Rulli, L. Proietti and G. Lombardi, Ericsson Fatme R&D Division, Nov. 1993, pp. 1127–1133.

"Data Exchange Adapter for Micro Channel/370" Software Patent Institute, 1995,1996.

"1394 200 Mb/s PHYsical Layer Transceiver," IBM Microelectronics, Product Data Sheet And Application Notes, Version 1.4, Mar. 14, 1996.

"IEEE 1394–1995 Triple Cable Transceiver/Arbiter," Texas Instruments TSB21LV03, Product Review, Revision 0.99, Mar. 19, 1996.

P1394 Standard For A High Performance Serial Bus, IEEE, 1995.

"The IEEE–1394 High Speed Serial Bus," R.H.J. Bloks, Philips Journal Of Research, vol. 50, No. 1/2, pp. 209–216, 1996.

"PC Intern 4 Systemprogrammierung," Michael Tischer, pp. 162–181, Data Becker GmbH, 1994, Dusseldorf, Germany.

* cited by examiner

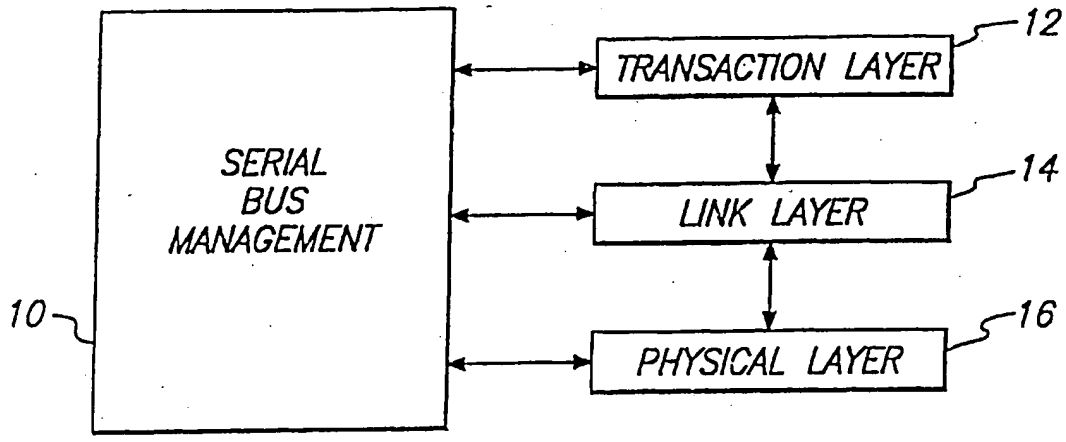


FIG. 1
(PRIOR ART)

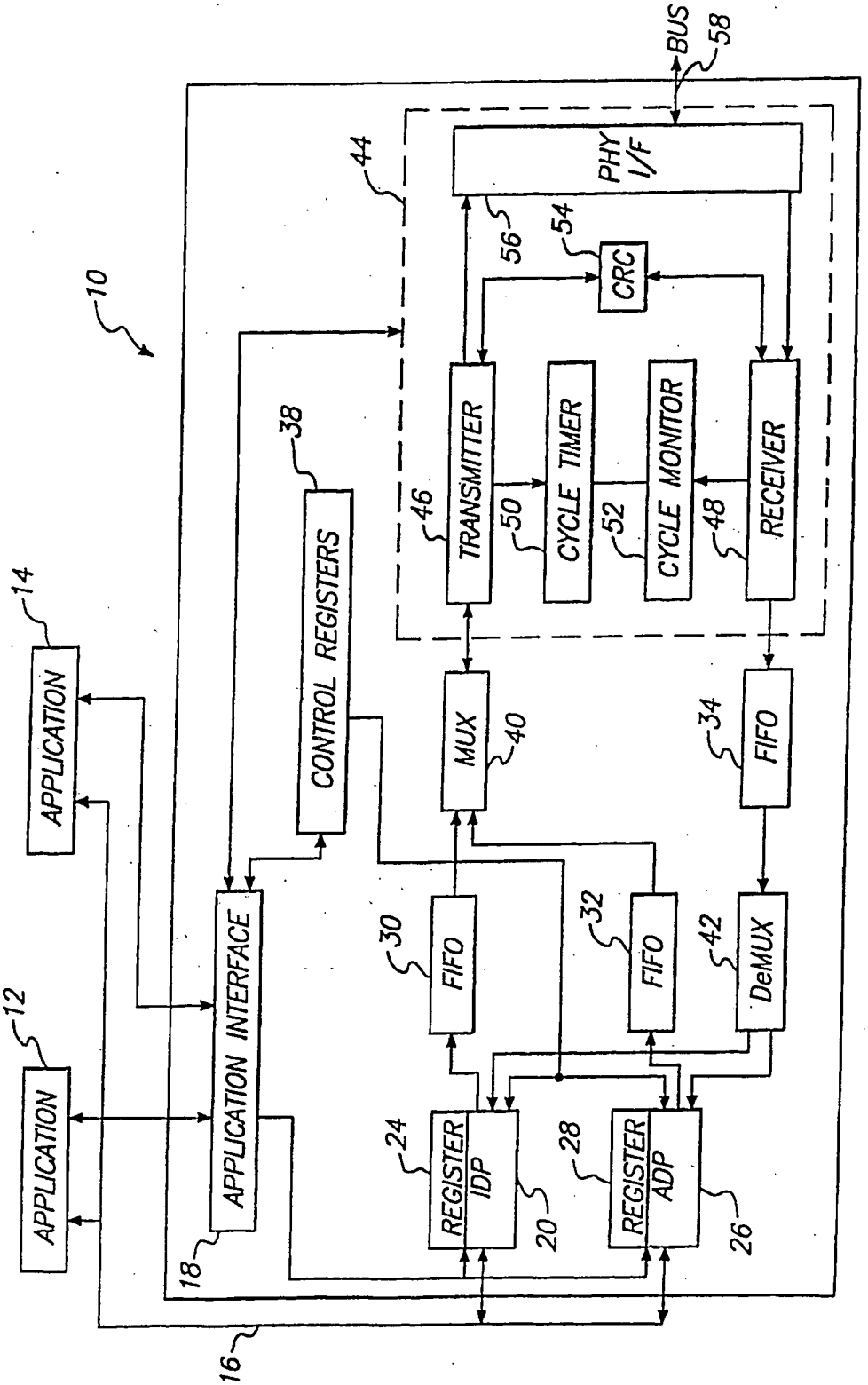

BUS
58
PHY I/F
44
56
54
CRC
10
38
TRANSMITTER
CYCLE TIMER
CYCLE MONITOR
RECEIVER
46
50
52
48
CONTROL REGISTERS
14
APPLICATION
MUX
40
34
FIFO
30
FIFO
32
FIFO
42
DeMUX
12
APPLICATION
APPLICATION INTERFACE
18
24
REGISTER IDP
20
28
REGISTER ADP
26
16

FIG. 2

OFFSET	R/W	FUNCTION			
		BYTE 0 (msb)	BYTE 1	BYTE 2	BYTE 3 (lsb)
0	RW	CYCLE TIME (20 BITS)			
4	RW	CONTROL			
8	RW	STATUS			
C					
10					
14					
18					
1C					
20	RW	pending_ch_mask_hi			
24	RW	pending_ch_mask_lo			
28	RO	ch_mask_hi			
2C	RO	ch_mask_lo			
30					
34	RW	CS_output			CS_addr_CS
38	RW	CS_addr_0			
3C	RW				
40	RW				
44	RW				
48	RW				
4C	RW				
50	RW				
54	RW				
58	RW				
5C	RW				
60	RW				
64	RW				
68	RW				
6C	RW				
70	RW				
74	RW				CS_addr_63

IDP Register File

FIG. 3

90

NAME	REG CODE	WIDTH	SRC/DEST?	REGISTER USAGE
TMM	0	34	SRC	IMMEDIATE VALUE
BUS_IN	1	32	SRC	RECEIVE FIFO
BUS_OUT	2	34	DEST	OUTBOUND ISOCHRONOUS FIFO
	3			
D0	4	34	SRC/DEST	DATA REGISTER 0
D1	5	34	SRC/DEST	DATA REGISTER 1
D2	6	34	SRC/DEST	DATA REGISTER 2
D3	7	34	SRC/DEST	DATA REGISTER 3
D4	8	34	SRC/DEST	DATA REGISTER 4
D5	9	34	SRC/DEST	DATA REGISTER 5
D6	A	34	SRC/DEST	DATA REGISTER 6
D7	B	34	SRC/DEST	DATA REGISTER 7
	C			
	D			
	E			
	F			
DATA_0	10	32	SRC/DEST	DATA INTERFACE DMA CHANNEL 0
DATA_1	11	32	SRC/DEST	DATA INTERFACE DMA CHANNEL 1
DATA_2	12	32	SRC/DEST	DATA INTERFACE DMA CHANNEL 2
DATA_3	13	32	SRC/DEST	DATA INTERFACE DMA CHANNEL 3
	14			
	15			
	16			
	17			
	18			
	19			
	1A			
	1B			
	1C			
	1D			
	1E			
	1F			

FIG. 4

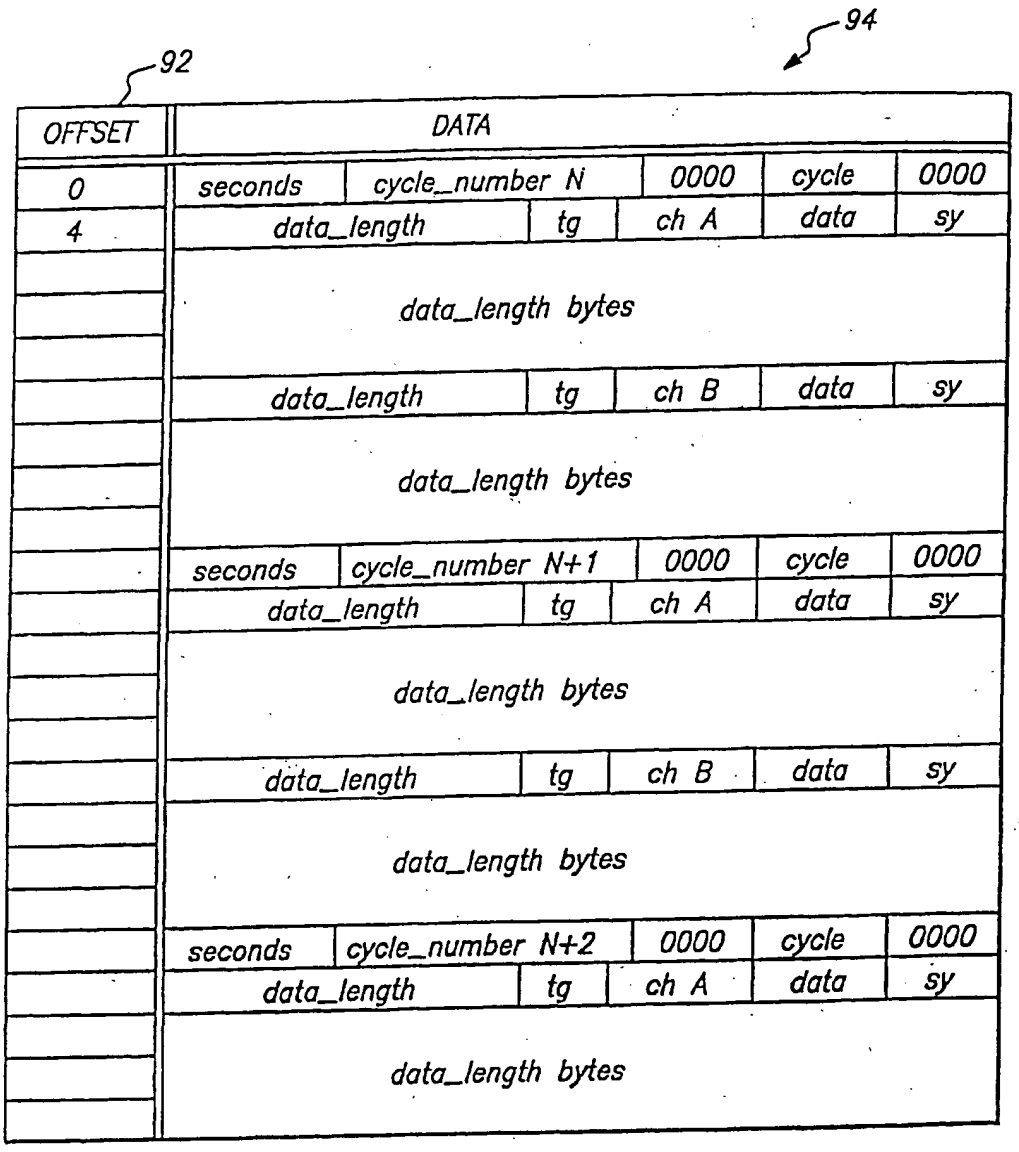

92
94
OFFSET
DATA
0
4
seconds
cycle_number N
0000
cycle
0000
data_length
tg
ch A
data
sy
data_length bytes
data_length
tg
ch B
data
sy
data_length bytes
seconds
cycle_number N+1
0000
cycle
0000
data_length
tg
ch A
data
sy
data_length bytes
data_length
tg
ch B
data
sy
data_length bytes
seconds
cycle_number N+2
0000
cycle
0000
data_length
tg
ch A
data
sy
data_length bytes

FIG. 5

ISOCHRONOUS DATA PIPE FOR MANAGING AND MANIPULATING A HIGH-SPEED STREAM OF ISOCHRONOUS DATA FLOWING BETWEEN AN APPLICATION AND A BUS STRUCTURE

FIELD OF THE INVENTION

The present invention relates to the field of conducting isochronous data transfer operations to and from an application over a bus structure. More particularly, the present invention relates to the field of managing and manipulating a high-speed stream of isochronous data to complete a data transfer operation between an application and node coupled to a bus structure.

BACKGROUND OF THE INVENTION

The IEEE 1394 standard, "P1394 Standard For A High Performance Serial Bus," Draft 8.01v1, Jun. 16, 1995, is an international standard for implementing an inexpensive high-speed serial bus architecture which supports both asynchronous and isochronous format data transfers. Isochronous data transfers are real-time transfers which take place such that the time intervals between significant instances have the same duration at both the transmitting and receiving applications. Each packet of data transferred isochronously is transferred in its own time period. An example of an ideal application for the transfer of data isochronously would be from a video recorder to a television set. The video recorder records images and sounds and saves the data in discrete chunks or packets. The video recorder then transfers each packet, representing the image and sound recorded over a limited time period, during that time period, for display by the television set. The IEEE 1394 standard bus architecture provides multiple channels for isochronous data transfer between applications. A six bit channel number is broadcast with the data to ensure reception by the appropriate application. This allows multiple applications to simultaneously transmit isochronous data across the bus structure. Asynchronous transfers are traditional data transfer operations which take place as soon as possible and transfer an amount of data from a source to a destination.

The IEEE 1394 standard provides a high-speed serial bus for interconnecting digital devices thereby providing a universal I/O connection. The IEEE 1394 standard defines a digital interface for the applications thereby eliminating the need for an application to convert digital data to analog data before it is transmitted across the bus. Correspondingly, a receiving application will receive digital data from the bus, not analog data, and will therefore not be required to convert analog data to digital data. The cable required by the IEEE 1394 standard is very thin in size compared to other bulkier cables used to connect such devices. Devices can be added and removed from an IEEE 1394 bus while the bus is active. If a device is so added or removed the bus will then automatically reconfigure itself for transmitting data between the then existing nodes. A node is considered a logical entity with a unique address on the bus structure. Each node provides an identification ROM, a standardized set of control registers and its own address space.

The IEEE 1394 standard defines a protocol as illustrated in FIG. 1. This protocol includes a serial bus management block **10** coupled to a transaction layer **12**, a link layer **14** and a physical layer **16**. The physical layer **16** provides the electrical and mechanical connection between a device or application and the IEEE 1394 cable. The physical layer **16** also provides arbitration to ensure that all devices coupled to the IEEE 1394 bus have access to the bus as well as actual data transmission and reception. The link layer **14** provides data packet delivery service for both asynchronous and isochronous data packet transport. This supports both asynchronous data transport, using an acknowledgement protocol, and isochronous data transport, providing real-time guaranteed bandwidth protocol for just-in-time data delivery. The transaction layer **12** supports the commands necessary to complete asynchronous data transfers, including read, write and lock. The serial bus management block **10** contains an isochronous resource manager for managing isochronous data transfers. The serial bus management block **10** also provides overall configuration control of the serial bus in the form of optimizing arbitration timing, guarantee of adequate electrical power for all devices on the bus, assignment of the cycle master, assignment of isochronous channel and bandwidth resources and basic notification of errors.

To initialize an isochronous transfer, several asynchronous data transfers may be required to configure the applications and to determine the specific channel which will be used for transmission of the data. Once the channel has been determined, buffers are used at the transmitting application to store the data before it is sent and at the receiving application to store the data before it is processed. In a general purpose host or peripheral implementation, the format of the transmitted data is not in a form which can be used by the application. In most cases, a general purpose processor must preprocess the stream of data before sending it to the application. Often, the preprocessing task consumes considerable computational power which can make it impossible to effectively handle the real time stream of data.

What is needed is an isochronous data pipe that provides the ability to the application to manage and manipulate a high-speed stream of data being sent from or received by the application over a bus structure. What is further needed is an isochronous data pipe which allows the application to transmit and receive data in its native format, thereby improving the ability of the application to effectively handle a continuous stream of data over time.

SUMMARY OF THE INVENTION

An isochronous data pipe provides a bi-directional path for data between an application and a bus structure. The isochronous data pipe includes the ability to send, receive and perform manipulations on any isochronous stream of data, including data on any number of isochronous channels. The isochronous data pipe is a programmable sequencer that operates on the stream of isochronous data as it passes through the isochronous data pipe. The isochronous data pipe is programmed by an application to perform specific operations on the stream of data before the data is either transmitted across the bus structure or sent to the application, thereby pre-processing and manipulating the data before it is delivered to its destination. The operations are performed on both the packet header and the data field of the data packet. The isochronous data pipe can be stopped and started on the occurrence of specific events. In an alternate embodiment of the present invention, the isochronous data pipe is programmed to send and receive both isochronous and asynchronous data, including generating requests and appropriate packet headers.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a protocol defined by the IEEE 1394 standard.

FIG. 2 illustrates a block diagram schematic of a link circuit including an isochronous data pipe according to the present invention and an asynchronous data pipe.

FIG. 3 illustrates a register file within the isochronous data pipe.

FIG. 4 illustrates a register file within the isochronous data pipe sequencer.

FIG. 5 illustrates an example of an isochronous data stream showing the isochronous recording format.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

An isochronous data pipe transmits and receives data for an application across a bus structure. Preferably, the bus structure is an IEEE 1394 standard bus structure. The isochronous data pipe is programmable and will execute a series of instructions on a stream of data in order to perform manipulations on the data required by the application. In a link circuit, an isochronous data pipe is included for transmitting and receiving isochronous data and an asynchronous data pipe is included for transmitting and receiving asynchronous data. The data from the isochronous data pipe and the asynchronous data pipe is multiplexed onto the bus structure. The data received from the bus structure is demultiplexed to the isochronous data pipe and the asynchronous data pipe. Alternatively, the isochronous data pipe is programmed to transmit and receive both isochronous and asynchronous data.

A link circuit including an isochronous data pipe (IDP), according to the present invention, and an asynchronous data pipe is illustrated in FIG. 2. The link circuit **10** provides a link between applications **12** and **14** and a bus structure **58**. The applications **12** and **14** are both coupled to a system bus **16**. The system bus **16** is coupled to both the isochronous data pipe **20** and the asynchronous data pipe **26**. The applications **12** and **14** are also both coupled to an applications interface circuit **18**. The applications interface circuit **18** is coupled to a set of control registers **38**, to the isochronous data pipe **20**, to the asynchronous data pipe **26** and to a link core **44**. Both the isochronous data pipe and the asynchronous data pipe **26** include a register set **24** and **28**, respectively. The outbound FIFO **30** corresponds to the isochronous data pipe **20** and is coupled between the isochronous data pipe **20** and a multiplexer **40**. The outbound FIFO **32** corresponds to the asynchronous data pipe **26** and is coupled between the asynchronous data pipe **26** and the multiplexer **40**. The control registers **38** are also coupled to both the isochronous data pipe **20** and the asynchronous data pipe **26**. An inbound FIFO **34** is coupled to a demultiplexer **42**. The demultiplexer **42** is coupled to both the isochronous data pipe **20** and the asynchronous data pipe **26**.

The link core **44** includes a transmitter **46**, a receiver **48**, a cycle timer **50**, a cycle monitor **52**, a CRC error checking circuit **54** and a physical interface circuit **56** for physically interfacing to the bus structure **58**. The transmitter **46** is coupled to the multiplexer **40**, to the cycle timer **50**, to the CRC error checking circuit **54** and to the physical interface circuit **56**. The receiver **48** is coupled to the inbound FIFO **34**, to the cycle monitor **52**, to the CRC error checking circuit **54** and to the physical interface circuit **56**. The cycle timer **50** is coupled to the cycle monitor **52**. The physical interface circuit **56** is coupled to the bus structure **58**.

The link circuit **10**, illustrated in FIG. **2**, includes a single FIFO **34** for all incoming data, both isochronous and asynchronous, a FIFO **30**, dedicated to the isochronous data pipe **20** for outbound data and a FIFO **32**, dedicated to the asynchronous data pipe **26** for outbound data. The outbound data from the FIFOs **30** and **32** are multiplexed, by the multiplexer **40**, through the link core **44** and onto the bus structure **58**. The inbound data from the FIFO **34** is directed to either the isochronous data pipe **20** or the asynchronous data pipe **26**, by the demultiplexer **42**, as will be discussed below.

Preferably, the inbound FIFO **34** is thirty-three bits wide, the outbound FIFO **30** is thirty-four bits wide and the outbound FIFO **32** is thirty-three bits wide. In each of the FIFOs **30**, **32** and **34**, bits **0** through **31** are designated to carry data and bit **32** is designated to carry a packet boundary marker. For outbound packets, the isochronous data pipe **20** and the asynchronous data pipe **26** set the bit **32** to a logical high voltage level on the first quadlet of each packet. For inbound packets, the link core **44** sets the bit **32** to a logical high voltage level on the first quadlet of each packet.

In the outbound FIFO **30**, corresponding to the isochronous data pipe **20**, bit **33** is designated to indicate an isochronous cycle boundary. The isochronous data pipe **20** sets the bit **33** to a logical high voltage level on the first quadlet of the first isochronous packet in each isochronous cycle. When the link core **44** receives a quadlet of data with the bit **33** set to a logical high voltage level, it delays until the next cycle start, then transmits all isochronous packets in the outbound FIFO **30** until another quadlet with the bit **33** set to a logical high voltage level is detected.

To transmit application data, from one of the applications **12** and **14**, onto the bus structure **58**, the isochronous data pipe **20** and the asynchronous data pipe **26** both generate appropriate header information and append the appropriate application data to form a packet in the form required by the bus structure **58**. These packets are then stored in the appropriate FIFO **30** and **32** for transmission onto the bus structure **58**.

The asynchronous data pipe **26** is preferably implemented as described in co-pending U.S. patent application Ser. No. 08/612,321, filed on the same date as the present application and entitled "Asynchronous Data Pipe For Automatically Managing Asynchronous Data Transfers Between An Application And A Bus Structure," which is hereby incorporated by reference. The asynchronous data pipe **26** automatically generates transactions necessary to complete asynchronous data transfer operations for an application over a bus structure. The asynchronous data pipe **26** includes a register file **28** which is programmed by the application. The register file **28** allows the application to program requirements and characteristics for the data transfer operation. The register file **28** includes bus speed, transaction label, transaction code, destination node identifier, destination offset address, length of each data packet, packet counter, packet counter bump field, control field and a status field.

After the register file **28** is programmed and initiated by the application, the asynchronous data pipe **26** automatically generates the read or write transactions necessary to complete the data transfer operation over the appropriate range of addresses, using the information in the register file as a template for generating the transactions and headers. The asynchronous data pipe **26** automatically increments the value in the destination offset address field for each transaction according to the length of each data packet, unless an incrementing feature has been disabled, signalling that the transactions are to take place at a single address. The packet counter value represents the number of transactions remaining to be generated. The packet counter value is decre-

mented after each packet of data is transferred. The packet counter bump field allows the application to increment the packet counter value by writing to the packet counter bump field.

Multiple asynchronous data pipes can be included within a link circuit **10** for managing multiple asynchronous data transfer operations. In such a system, each asynchronous data pipe has its own unique transaction label value or range of values. The multiplexer **40** multiplexes the transactions and data packets from the asynchronous data pipes and the isochronous data pipe onto the bus structure **58**. The demultiplexer **42** receives signals and data packets from the bus structure **58** and routes them to the appropriate asynchronous data pipe or isochronous data pipe, using the transaction code and the transaction label values.

In the link circuit **10** there is only one isochronous data pipe **20**. This isochronous data pipe **20** can handle multiple isochronous channels and at the data interface, the isochronous data pipe **20** can interact with more than one application. Therefore, the isochronous data pipe **20** can support more than one stream of isochronous data, where each stream of data is made up of one or more isochronous channels. In an alternative embodiment, as will be described below, the isochronous data pipe **20** can also send and receive asynchronous data, thereby performing the functions of an asynchronous data pipe.

The link core **44** accepts packets of data from the outbound FIFOs **30** and **32**, creates packets which comply with the format required by the bus structure **58** and then transfers the packets through the physical interface **56** onto the bus structure **58**. The link core **44** transmits one isochronous cycle's worth of data from the outbound isochronous FIFO **30** on each isochronous cycle. When not transmitting isochronous data, the link core **44** transmits asynchronous packets from the outbound asynchronous FIFO **32**.

The link core **44** transmits all received packets to the inbound FIFO **34**. Unless the link core **44** is operating in a snoop mode, the link core **44** only receives asynchronous packets addressed to the appropriate node ID and isochronous packets with the proper channel numbers. In the snoop mode, the link core **44** receives all packets regardless of their destination node ID or isochronous channel number.

The isochronous data pipe **20** provides a bi-directional data path for application data which is to be transmitted over the bus structure **58**. A stream of isochronous data is made up of data on one or more isochronous channels. The isochronous data pipe **20** can operate on any arbitrary stream of isochronous data, containing data on any number of isochronous channels. The isochronous data pipe **20** is a programmable sequencer that operates on a stream of isochronous data from the bus **16** to the outbound isochronous FIFO **30** or from the receive FIFO **34** to the bus **16**.

For each quadlet of data transferred, the isochronous data pipe **20** executes a predetermined number of instructions to manipulate the data as necessary. These instructions can operate on the isochronous data block packet. When sending data to be output on the bus structure **58**, the stream of data output by the isochronous data pipe **20**, is dependent on both the stream of data input to the isochronous data pipe **20** and the manipulations performed on the data by the isochronous data pipe **20**. Correspondingly, when receiving data from the bus structure **58**, the stream of data output by the isochronous data pipe **20** on the bus **16**, is dependent on the stream of data input to the isochronous data pipe **20** and the manipulations performed on the data by the isochronous data pipe **20**.

The isochronous data pipe **20** supports several scheduling features for the starting and stopping of isochronous data transfers, depending on the current mode of operation of the isochronous data pipe. With proper programming, the isochronous data pipe supports the isochronous recording data formats, as defined in the SCSI-3 Serial Bus Protocol standard. This protocol defines how to label an isochronous stream of data when it is recorded so that it can be recreated precisely when played back. The isochronous data pipe is a programmable data handling engine in the isochronous data path. With proper programming, this engine implements the isochronous recording formats, plus includes the ability to filter the data by deleting quadlets, or performing specific operations on each quadlet transferred to or from the bus structure **58**.

The FIFO interface for both the isochronous data pipe **20** and the asynchronous data pipe **26** is coupled directly to a FIFO **30** and **32**, respectively. The FIFO **30** is dedicated to the data path controlled by the isochronous data pipe **20**. The FIFO **32** is dedicated to the data path controlled by the asynchronous data pipe **26**. The link interface for the isochronous data pipe **20** and the asynchronous data pipe **26** are both coupled through the multiplexer **40** and the demultiplexer **42** to the link core **44**. The data presented from the isochronous data pipe **20** and the asynchronous data pipe **26** to the link core **44** is in a format required by the link core function. Both the isochronous data pipe **20** and the asynchronous data pipe **26** expect the data coming from the link core **44** to be in the format defined by the link core specification. If additional logical blocks are included within a system, each logical block is coupled to the link core **44** through the multiplexer **40** and the demultiplexer **42**. For example, multiple asynchronous data pipes could be included within a system. In a system with multiple asynchronous data pipes, each of the asynchronous data pipes are coupled to the multiplexer **40** through the FIFO **32**. In such a system, an additional multiplexer is included between the asynchronous data pipes and the FIFO **32** for multiplexing packets of data into the FIFO **32**.

When directing data from the isochronous data pipe **20**, the multiplexer **40** recognizes that when data is available from the isochronous data pipe, the multiplexer **40** transmits one packet of data per isochronous cycle per channel. The data sent from the link core **44** to the isochronous data pipe **20** and the asynchronous data pipe **26** is routed through the FIFO **34** and the demultiplexer **42**. The demultiplexer **42** does not change any information when it routes packets from the link core **44** to the appropriate one of the isochronous data pipe **20** or the asynchronous data pipe **26**. All information produced by the link core is sent to the destination logical block. The isochronous data pipe **20** and the asynchronous data pipe **26** will perform all necessary manipulation of the data from the link core **44** before this data is transferred to one of the applications **12** and **14**, which may include stripping header information required by the protocol for the bus structure **58**. For outbound data, the isochronous data pipe **20** and the asynchronous data pipe **26** both prepare data from the application so that it is in the proper form, as required by the link core **44**. Both the isochronous data pipe **20** and the asynchronous data pipe **26** will generate the appropriate header information and embed that in the data from the application before sending the data to the link core **44** through the muliplexer **40**.

For both the isochronous data pipe **20** and the asynchronous data pipe **26**, the link interface produces and consumes data in a format which is compatible with the requirements of the link core **44** function. During a data send operation,

the isochronous data pipe **20** will generate the required bus structure specific header information and embed it in the data from the application, as required by the link core **44**. During a data receive operation, for data moving from the link core **44** to either the isochronous data pipe **20** or the asynchronous data pipe **26**, the isochronous data pipe **20** and the asynchronous data pipe **26** both accept that data in the format provided by the link core **44**. In other words, no manipulation of the data is required to translate data from the link core **44** to the isochronous data pipe **20** or the asynchronous data pipe **26**.

When only one logical block is included within a system, that logical block can be connected directly to the link core **44**. When there are multiple logical blocks within a system, the system includes an appropriate multiplexer **40** and demultiplexer **42** between the logical blocks and the link core **44**. The multiplexer **40** is responsible for taking the data at the link interfaces of the multiple logical blocks and multiplexing that data through the link core **44** and onto the bus structure **58** on a packet by packet basis. This information is application specific and is routed to the bus structure in a priority set by the transferring operation. Each isochronous data packet is sent by the multiplexer **40** during its appropriate time period. The demultiplexer **42** uses the value in the transaction code and the channel number fields of each packet received from the bus structure **58** to route the packet to the appropriate logical block **20** or **26**. If there is no more than one isochronous data pipe **20** and one asynchronous data pipe **26**, then the transaction code is all that is required to route the packet appropriately. The demultiplexer **42** will first read the transaction code to determine that the packet is asynchronous data and should be routed to an asynchronous data pipe. If there is more than one asynchronous data pipe within the system, the demultiplexer **42** then uses the value in the transaction label of the asynchronous response packet header to route the packet to the proper asynchronous data pipe.

The isochronous data pipe of the present invention is a bidirectional data path between a corresponding FIFO and the link-core **44**. With proper programming, the isochronous data pipe supports the isochronous data recording format, as documented in the SCSI-3 Serial Bus Protocol (SBP) standard and allows programmable manipulation of the data in the isochronous stream.

When transferring data through the corresponding FIFO **30** to the link core **44** or when receiving data from the demultiplexer **42**, the isochronous data pipe **20** operates on each quadlet of data independently. The isochronous data pipe **20** performs a programmable number of instructions on each quadlet in order to manipulate the data, as necessary. The possible instructions which can be performed by the isochronous data pipe **20** are included within an instruction set, which will be discussed in detail below. The isochronous data pipe **20** also includes an independent, dedicated register file **24** which will also be discussed in detail below.

If a bus reset occurs while the isochronous data pipe **20** is transferring data, the isochronous data pipe **20** operation resumes exactly where it left off when the next cycle start packet appears on the bus structure **58**. Note that although the processing of isochronous data resumes immediately, the embedded application reallocates any channel numbers, bandwidth and any connections in use prior to the bus reset, as defined in the IEEE 1394 standard and the IEC standard for consumer devices.

The isochronous data pipe **20** is controlled by an independent, dedicated register file, as illustrated in FIG. **3**. This register file is programmed by the originating application and used to generate headers, instructions and transactions necessary to complete an isochronous data transfer operation across the bus structure **58**. The register file **80** includes 120 bytes of data, numbered hexadecimally 0 through 77. In FIG. **3**, the register file **80** is illustrated in a table format with 30 horizontal rows, each including four bytes of data. An offset column **82** is included in FIG. **3**, to show the offset of the beginning byte in each row from the address of the beginning of the register file **80**. A read/write column **84** is also included to show whether the fields in each row can be either read from and written to or read from only.

The cycle time field cycle_time is a twenty bit field within bytes **0–2** of the register file **80**. The cycle time field can be read from and written to. When the control event field, which will be discussed below, contains the cycle number value, the cycle time field holds the cycle time on which the isochronous data pipe **20** will start or stop transferring isochronous data.

The control field is a thirty-two bit field within bytes **4–7** of the register file **80**. The control field can be read from and written to. The control field includes an event field, an output enable field, a stop on error field, a transmit enable field and a go field. The event field is a four bit field in bits **28–31** of the control register. The value in the event field defines the bus event for the isochronous data pipe **20** to use as a trigger. When this bus event occurs, the isochronous data pipe transfers the value stored in the pending channel mask register pending_ch_mask to the current channel mask register ch_mask. The event field is encoded for the possible bus events as illustrated in Table I below.

TABLE I

value	meaning
0	immediately
1	cycle number
2	reserved
3	reserved
4-F	reserved

Therefore, when the event field holds a value equal to 0, the isochronous data pipe will then start or stop immediately. When the event field holds a value equal to 1, the isochronous data pipe will then start or stop, as specified by the value in the cycle time field, as discussed above.

The output enable field is a four bit field in bits **4–7** of the control field. When any of the bits in the output enable field are set to a logical high voltage level, then the corresponding DMA channel will assure that the prefill FIFO is kept full and the isochronous data pipe **20** will dispatch to the control store output instruction whenever there is an empty quadlet in the outbound FIFO **30**.

The stop on error field is a one bit field in bit **3** of the control field. When the stop on error bit is set to a logical high voltage level, the isochronous data pipe **20** will stop the current operation on the first error encountered by setting the value in the channel mask register to a logical low voltage level. Possible errors when sending data include a FIFO underrun or a missing cycle start packet. Possible errors when receiving data include a FIFO overrun, a missing cycle start packet, a data CRC error, an error in packet format or a channel missing error.

The transmit enable field is a one bit field in bit **1** of the control field. When the transmit enable bit is set to a logical high voltage level, the isochronous data pipe **20** will begin

executing the output control store program. When the go bit is at a logical low voltage level or the output control store program executes a return instruction, the transmit enable bit will be cleared.

The go field is a one bit field in bit 0 of the control field. The application sets the go bit to a logical high voltage level to enable the isochronous data pipe to watch for an event. When the specified event condition is satisfied, the isochronous data pipe **20** transfers the contents of the pending channel mask register to the current channel mask register.

The status field is a thirty-two bit field within bytes 8-B of the register file **80**. The status field can be read from and written to. The status field contains status information which reports the current state of the isochronous data pipe **20**. The bits **0–7** of the status field correspond to the bits **0–7** of the control field and include an output field, a stop on error field, a transmit enable field and an active field. The value of these fields in the status register indicate the current operational state of the isochronous data pipe **20**. The bits **8–27** of the status field are reserved. Within the status field, the active field is a one bit field in bit **0** of the status field, which indicates whether or not the isochronous data pipe is active. Preferably, if the active bit is equal to a logical high voltage level, the isochronous data pipe is currently active and transferring data. If the active bit is equal to a logical low voltage level, the isochronous data pipe is not currently active. The error field is a four bit field in bits **28–31** of the status field. When the isochronous data pipe **20** halts operation due to an error, the error field contains a value indicating the error condition. The error field is only valid when the active bit is equal to a logical low voltage level. The possible values for the error field and the error to which they correspond are listed in Table II below.

TABLE II

Value	Error
0	FIFO overrun
1	HFO underrun
2	Missing cycle start packet
3	Data CRC error
4	Missing cycle start packet
5	Error in packet format

The pending channel mask high field pending_ch_mask_hi is a four byte field within bytes **20–23** of the register file **80**. The pending channel mask low field pending_ch_mask_lo is a four byte field within bytes 24–27 of the register file **80**. Together, the two pending channel mask fields pending_ch_mask_hi and pending_ch_mask_lo form an eight byte field containing the mask of isochronous channel numbers for the isochronous data pipe **20** to receive. The isochronous data pipe **20** transfers the contents of this field to the channel mask register when the programmed trigger event occurs. The bit assignment of the pending channel mask field is the same as the bit assignment of the channels available register defined in chapter eight of the IEEE 1394 standard.

The current channel mask high field ch_mask_hi is a four byte field within bytes **28–2B** of the register file **80**. The current channel mask low field ch_mask_lo is a four byte field within bytes 2C–2F of the register file **80**. Together, the two current channel mask fields ch_mask_hi and ch_mask_lo form an eight byte field containing the channel mask currently in operation, with each bit within the current channel mask fields representing an isochronous channel. The channel mask field is only loaded from the pending channel mask field when a trigger event occurs. The isochronous data pipe **20** ignores received isochronous channel numbers for which the corresponding bit in the current channel mask field is set to a logical high voltage level.

The control store output field CS_output is a one byte field within byte **34** of the register file **80**. The control store output field CS_output contains the control store address within the control store memory, to which the isochronous data pipe **20** dispatches whenever there is an empty quadlet in the outbound FIFO **30** and the isochronous data pipe **20** is not currently receiving an isochronous packet of data. The control store memory contains instructions used by the isochronous data pipe in performing its operations on a stream of data.

The control store cycle start address field CS_addr_CS is a one byte field within byte **37** of the register file **80**. The control store address field CS_addr_CS contains the control store address to which the isochronous data pipe branches when the cycle start packet is received. The first quadlet available to the control store program is the first quadlet of the cycle start packet. The control store address fields CS_addr_0 through CS_addr_63 are each one byte fields within bytes **38** through **77** of the register file **80**. These fields contain the control store address store where the isochronous data pipe is to branch upon receiving data on the isochronous channel matching the byte number of the control store address field. For example, the control store address field CS_addr_10 contains the address in the control store where the isochronous data pipe is to branch upon receiving data on the isochronous channel number **10**. The isochronous data pipe ignores all isochronous channels for which the corresponding value in the control store address field is equal to FFh. It should be noted that the behavior of the control store address field is the same when transmitting as when receiving isochronous data.

There are sixty-four potential isochronous channels **0–63**. The control store address fields CS_addr_0 through CS_addr_63 each correspond to an isochronous channel and contain the address in the control store memory where the instructions for that isochronous channel begin. Accordingly, when the isochronous data pipe **20** receives data on a particular isochronous channel, the isochronous data pipe **20** branches to the address contained in the corresponding control store address field to obtain the instructions for manipulating the data for that channel. Isochronous channels for which the corresponding value in the control store address field is equal to FFh are ignored.

A stream of isochronous data is made up of one or more isochronous channels. The isochronous data pipe **20** receives isochronous channels for which the corresponding bit in the current channel mask field is set to a logical high voltage level. The isochronous data pipe **20** transmits isochronous data according to the control store program beginning at the control store address pointed to by the value in the control store output CS_output register. For example, if the isochronous channels **3, 4** and **5** exist on the bus structure **58** and the application wants the isochronous data pipe **20** to combine channels **3** and **5** into a single stream without performing any manipulation on the data contained in these isochronous channels, then the application programs a value of "10h," for example, into the control store address fields CS_addr_3 and CS_addr_5. At the control store address "10h," the application then loads an instruction sequence as shown in Table III.

TABLE III

```
        SHIFTI  BUS_IN, 16, D0   ;Shift to get the data length
                                  value
        ADDI    D0, 3, D0       ;Wrap up
        ANDI    D0, FFFC, D0    ;and mask to get count plus pad
        BZ      HALT            ;Done if data length equals zero
CONT:   MOVE    BUS_IN, DATA_0  ;Move a data word to DMA
                                  channel 0
        SUBI    D0, 4, D0       ;Decrement byte count
        BNZ     CONT            ;Continue if not zero
HALT:   RET                     ;Finished with this packet
```

The application then programs a value of "28h" into the pending channel mask, then writes a value of "1" into the control field. This value in the control field indicates an event of immediate with no DMA channels programmed for output. The result is that the isochronous data pipe **20** immediately shifts the value of the pending channel mask field into the current channel mask field. Because the bits **3** and **5** are now set to a logical high voltage level in the current channel mask field, the isochronous data pipe **20** will begin processing the isochronous channels **3** and **5** according to the control store program beginning at address "10h." Note that in this example both the control store fields CS_addr_3 and CS_addr_5 contain a value of "10h," so that the data for both of the isochronous channels **3** and **5** is processed according to the same control store instruction sequence, beginning at the address "10h."

The control store program illustrated in Table III is a program which moves the data from the receive FIFO **34** onto the DMA channel **0** on the bus **16**. The isochronous data pipe **20** ignores any data received on isochronous channel **4** because the bit **4** in the current channel mask field is not set to a logical high voltage level.

In the last line of the control store program illustrated in Table III, a return instruction is included. In all cases, the return instruction causes the isochronous data pipe **20** to perform the same tasks; namely, the isochronous data pipe decrements the value of the stack pointer and dispatches to the instruction within the stack which the stack pointer is currently pointing to. If the stack pointer is equal to zero when a return instruction is executed, the isochronous data pipe **20** halts operation until the next enabled isochronous channel is received or a cycle start packet is received. If the isochronous data pipe **20** is executing an output control store program, a return instruction will cause the isochronous data pipe to resume operation at the instruction where the output program was interrupted by the received isochronous packet.

The isochronous data pipe is actually a programmable sequencer which can be programmed to perform operations on the received stream of isochronous data. The isochronous data pipe sequencer contains a register file as illustrated in FIG. **4**. Within the register file **90**, the immediate value register IMM is a thirty-four bit register with a register code of "0" which can only be a source register. The immediate value register IMM specifies that the thirty-four bit immediate field of the instruction contains the source data for the given operation.

The bus input register BUS_IN is a thirty-two bit register with a register code of "1" which can only be a source register. Accessing the bus input register BUS_IN as a source of an operation clocks one quadlet of data from the receive FIFO **34** through the isochronous data pipe **20**. Subsequent accesses to the bus input register BUS_IN access subsequent quadlets of data in the input data stream.

The bus output register BUS_OUT is a thirty-four bit register with a register code of "2" which can only be a destination register. Accessing the bus output register BUS_OUT as a destination of an operation clocks one quadlet of data through the isochronous data pipe **20** to the outbound isochronous FIFO **30**. Subsequent accesses to the bus output register BUS_OUT clock subsequent quadlets of data in the output data stream.

The data registers D0–D7 are each thirty-four bit registers with a register code of "4", "5", "6", "7", "8", "9", "A" and "B", respectively, which can be either a source or destination register. The data registers D0–D7 can be used as the source or destination register for any operation.

The data interface registers DATA_0–DATA_3 are each thirty-two bit registers with a register code of "10", "11", "12" and "13", respectively, which can be either a source or destination register. Each of the data interface registers DATA_0–DATA_3 access a different DMA channel. Use of these registers is to be consistent with the programming of the output enable field DMA_out_en.

The isochronous data pipe **20** implements a stack made up of a linear list of eight one byte registers. The stack registers are only accessed during a branch to subroutine instruction and a return instruction. In the preferred embodiment of the present invention, the stack registers S0–S7, each have a respective register address 0–7. Alternatively, the actual number of stack registers will vary depending on the specific implementation. When the control store program is loaded, the stack pointer is automatically initialized to a value of zero, thereby pointing to the corresponding stack register S0.

When the isochronous data pipe **20** branches to a subroutine, the isochronous data pipe **20** decrements the stack pointer, stores the address of the next control store instruction into the current stack register, increments the value of the stack pointer, then branches to the control store instruction contained in the low order byte of the source field. When the isochronous data pipe **20** executes a return instruction, it decrements the stack pointer, then the isochronous data pipe **20** branches to the control store instruction contained in the current stack register. If the stack pointer is decremented when it contains a value equal to zero, the value of the stack pointer will remain at zero and the isochronous data pipe **20** will halt operation until it receives an isochronous data packet or cycle start packet. When the isochronous data pipe is executing an output control store program and a cycle start packet or enabled isochronous channel is received, the isochronous data pipe **20** will interrupt execution of the output control store program, save the address of the current instruction in the stack, decrement the stack pointer and then dispatch to the proper location to handle the received packet.

Each isochronous control store instruction includes an OpCode field, a source field, a destination field, an immediate value field, an immediate field and a reserved field. The OpCode field is a six bit field which describes an operation to perform, as will be discussed below in reference to FIG. **5**. The source field src is a four bit field which specifies a register or immediate value which contains the source value for the specified operation. The destination field dest is a four bit field which specifies a destination register for the specified operation. The immediate value field imm_val is a one bit field which when set to a logical high voltage level, specifies that one of the operands is contained in the immediate field. The immediate field imm is a thirty-four bit field which specifies an immediate value to use for an operation if the immediate value field imm_val is set to a logical high voltage level. In the preferred embodiment of the present invention, the reserved field includes thirteen bits which are reserved for use in alternate embodiments of the isochronous data pipe **20**.

The operation codes which are implemented by the isochronous data pipe sequencer during manipulation of a data stream and can be included in the OpCode field are listed in Table IV below. The isochronous data pipe **20** will store the results for any of these operations into any register which is capable of being a destination, as illustrated in FIG. **4**, including the data registers D0–D7, the outbound isochronous FIFO **30** and any DMA channel which is configured as a destination.

TABLE IV

Name	Mnemonic	Value (HEX)	Function
MOVE	MOVE MOVEI	0	moves value in src register to dest register
MOVE Multiple	MOVEM	1	moves a block of quadlets between the source and destination (i.e., between a DMA register and the outbound FIFO)
		2	
		3	
AND	AND ANDI	4	ANDs the value in the src register to the immediate value or the value in the dest register, and stores the result into the dest register
OR	OR ORI	5	ORs the value in the src register to the immediate value or the value in the dest register, and stores the result into the dest register
SHIFT	SHIFT SHIFTI	6	SHIFTS the value in the src register by the immediate value or the value in the dest register and stores the result into the dest register; positive values cause the isochronous data pipe to shift right; the isochronous data pipe fills the input bits with zeros
COMPARE	CMP CMPI	7	subtracts the immediate value from the value in the src register, or subtracts the value in the src register from the value in the dest register, but does not store the result; sets the Z bit according to the result of the subtraction
ADD	ADD ADDI	8	Adds value in src register to the immediate value or the value in dest register and stores the result in the dest register
SUBTRACT	SUB SUBI	9	Subtracts the immediate value from the value in the src register, or subtracts the value in the src register from the value in the dest register and stores the result in the dest register
MULTIPLY	MULT MULTI	A	Multiplies the immediate value by the value in the src register, or multiples the value in the src register by the value in the dest register and stores the result in the dest register
		B	
		C	
		D	
		E	
		F	
BRANCH	BRA	10	Branch to the control store address contained in the imm field
BRANCH ON ZERO	BZ	11	Branch to the control store address contained in the imm field if the result of the dest field from the previous operation was equal to zero
BRANCH ON NOT ZERO	BNZ	12	Branch to the control store address contained in the imm field if the result of the dest field from the previous operation was not equal to zero
		13	
BRANCH TO SUB	BSR	14	Decrement the stack pointer, save the address of the following instruction on the stack and branch to the CS address contained in the imm field
BRANCH TO SUB ON ZERO	BSRZ	15	If the result of the dest field from the previous operation was equal to zero, then decrement the stack pointer, save the address of the following instruction on the stack and branch to the CS address contained in the immediate field
BRANCH TO SUB ON NOT ZERO	BSRNZ	16	If the result of the dest field from the previous operation was not equal to zero, then decrement the stack pointer, save the address of the following instruction on the stack and branch to the CS address contained in the imm field
		17	
RETURN	RET	18	Branch to the instruction at the address contained on the stack; increment the stack pointer

For most of the operations listed in Table IV, there are included two mnemonic instructions. The mnemonic instruction which includes an "I" specifies the operation is to be conducted using the immediate value. The mnemonic instruction which does not include an "I" specifies the operation is to be conducted between the values in the source and destination registers.

When a MOVE operation is performed, the value in the register specified in the source field src is moved to the register specified in the destination field dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest

is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level.

When a MOVE Multiple operation is performed, a number of quadlets of data specified by a count value are moved from the register specified in the source field src to the register specified in the destination field dest. The count value is stored in the register designated in the immediate field of the instruction. Preferably, for the MOVE Multiple operation, the register specified in the source field src is one of the data interface registers DATA_0–DATA_3, which access a DMA channel, or the bus input register BUS_IN. Preferably, for this operation, the register specified in the destination field dest is one of the data interface registers DATA_0–DATA_3, which access a DMA channel, or the bus output register BUS_OUT.

During an AND operation, a logical AND operation is performed on the values in the source field src and the destination field dest and the result is stored in the register specified in the destination field dest. The ANDI form of this instruction uses the value in the immediate field instead of the value in the destination field as one of the operands and stores the result in the register specified in the destination field dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the AND operation is performed on all thirty-four bits.

During an OR operation, a logical OR operation is performed on the values in the registers specified by the source field src and the destination field dest and the result is stored in the register specified in the destination field dest. The ORI form of this instruction uses the value in the immediate field instead of the value in the destination field as one of the operands and stores the result in the register specified in the destination field dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the OR operation is performed on all thirty-four bits.

When a SHIFT operation is performed, the value in the destination register dest is shifted by the number of bits specified by the value in the source register src and the result is stored in the register specified in the destination field dest. A positive shift value shifts the value in the destination register to the right towards the least significant bit and zeros are used to fill in the shifted bits on the left beginning with the most significant bit. A negative shift value shifts the value in the destination register to the left towards the most significant bit and zeros are used to fill in the shifted bits on the right beginning with the least significant bit. The SHIFTI form of this instruction shifts the value in the source register by the number of bits specified in the immediate field and stores the result in the register specified in the destination field. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the shift operation is performed on only the low order thirty-two bits.

When a CMP operation is performed, the value in the source register src is subtracted from the value in the destination register dest. If the result of the CMP operation is a positive value, the Z bit is set to a logical high voltage level. If the result of the CMP operation is a negative or zero value, the Z bit is set to a logical low voltage level. The results of the CMP operation are not stored anywhere. The CMPI form of this instruction subtracts the immediate value from the value in the source register src, and sets the Z bit as specified above, according to the result. This instruction also does not store the result of the operation.

When an ADD operation is performed, the value in the source register src is added to the value in the destination register dest and the result is stored in the destination register dest. The ADDI form of this instruction adds the value in the source register src to the immediate value and stores the result in the destination register dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the ADD operation is performed on only the low order thirty-two bits.

When a SUB operation is performed, the value in the destination register dest is subtracted from the value in the source register src and the result is stored in the destination register dest. The SUBI form of this instruction subtracts the immediate value from the value in the source register and the result is stored in the destination register dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the SUB operation is performed on only the low order thirty-two bits.

When a MULT operation is performed, the value in the source register src is multiplied by the value in the destination register dest and the result is stored in the destination register dest. The MULTI form of this instruction multiplies the immediate value by the value in the source register src and the result is stored in the destination register dest. If the register specified in the source field src is a thirty-four bit register and the register specified in the destination field dest is a thirty-two bit register, the high order two bits will be lost. If the register specified in the source field src is a thirty-two bit register and the register specified in the destination field dest is a thirty-four bit register, then the high order two bits will both be set to a logical low voltage

level. If both the register specified in the source field src and the register specified in the destination field dest are thirty-four bit registers, then the MULT operation is performed on only the low order thirty-two bits.

When a BRANCH operation is performed, the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src. The source field src can specify a register or an immediate value.

When a BRANCH ON ZERO operation is performed, the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src if the result of the last arithmetic or move control store instruction was equal to zero. The source field src can specify a register or an immediate value.

When a BRANCH ON NOT ZERO operation is performed, the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src if the result of the last arithmetic or move control store instruction was not equal to zero. The source field can specify a register or an immediate value.

When a BSR operation is performed, the address of the next control store instruction is pushed onto the stack and the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src. The source field src can specify a register or an immediate value.

When a BSR ON ZERO operation is performed, if the result of the last arithmetic or move control store instruction was equal to zero, the address of the next control store instruction is pushed onto the stack and the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src. The source field src can specify a register or an immediate value.

When a BSR ON NOT ZERO operation is performed, if the result of the last arithmetic or move control store instruction was not equal to zero, the address of the next control store instruction is pushed onto the stack and the isochronous data pipe **20** branches to the control store address contained in the low order byte of the source field src. The source field src can specify a register or an immediate value.

When a RETURN operation is performed, the last control store address is popped off of the stack and the isochronous data pipe **20** branches to that address.

The isochronous recording format defined in the Serial Bus Protocol defines a standard format for recording a stream of isochronous data as transmitted over the bus structure **58**. The isochronous data pipe **20** of the present invention can be programmed to transform a received stream of isochronous data into the isochronous recording format, according to the Serial Bus Protocol. Correspondingly, the isochronous data pipe can also be programmed to create a stream of isochronous data from a stream of data in the isochronous recording format. A stream of data in the isochronous recording format is illustrated in FIG. **5**. It should be noted that the data stream illustrated in FIG. **5** begins on an isochronous cycle boundary.

In FIG. **5**, the data stream **94** includes data packets which are included for each isochronous cycle in both channels A and B. An offset column **92** is included in FIG. **5**, to show the offset of the beginning of each horizontal row. The header horizontal rows each include four bytes. The data section will include as many bytes as necessary to transfer the data packet. The header for each packet includes a seconds field, a cycle number field, and a cycle field. Each subheader for each channel within each packet includes a data_length field, a tag field tg, a channel field, a data field and a synchronizing field sy. The subheader is then followed by the data section within the packet.

EXAMPLE

Converting Isochronous Data To The Isochronous Recording Format

The control store program included in Table V below illustrates an example of how the isochronous data pipe **20** of the present invention can be programmed by an application to capture an isochronous stream of data consisting of channels **3** and **5**, map channel **3** to channel **7** and channel **5** to channel **9** and then send the resulting stream of data to DMA channel **0** in the isochronous recording format.

TABLE V

```
CS_addr_CS: ANDI  BUS_In, 0XFFFFF000, D3        ;mask cycle start packet
                  ORI   D3, 0Xcycle0, DATA_0    ;Send it to DMA ch 0
                  RET                           ;finished
CS_addr_5:        BSR   GET_QUAD                ;Get the isoch header
                  ORI   D2, 0X900, DATA_O       ;Map to channel 9 and output
                  BRA   GET_DATA                ;branch to get data field
CS_addr_3:        BSR   GET_QUAD                ;Get the isoch header
                  ORI   D2, 0X700, DATA_O       ;Map to channel 7 and output
GET_DATA:         SHIFTI D2, D16, D2            ;Get the data length
                  ADDI  D2, 3, D2               ;Wrap it up
                  ANDI  D2, FFFC, D2            ;and mask
                  BZ    HALT                    ;Finished if zero
MOVE_DATA:        MOVE  BUS_IN DATA_O           ;Get the next quadlet
                  SUBI  D2, 4, D2               ;Decrement quadlet counter
                  BNZ   MOVE_DATA               ;Continue if not zero
HALT:             RET                           ;Else, we're done
GET_QUAD:         ANDI  BUS_IN, 0XFFFFC00F, D2  ;Get the hdr w/o ch or tcode
                  ORI   D2, 0Xdata0, D2         ;Set the data marker
                  RET                           ;And return
```

EXAMPLE

Converting From Isochronous Recording Format

The control store program included in Table VI below illustrates an example of how the isochronous data pipe **20** of the present invention can be programmed by an application to take a stream of data at DMA channel **3** which is in the isochronous recording format and create a stream of isochronous data for transmission over the bus structure **58**. The source stream of data contains isochronous channels **7** and **9**. This control store program maps channel **7** to channel

1 and channel 9 to channel 2. Note that the program illustrated in Table VI requires that the first quadlet of data presented at DMA channel is a cycle start quadlet.

TABLE VI

```
CS_output:    MOVE   DATA_3 D0              ;Get a quadlet
              MOVEI        0x100000000, D3  ;Prepare the output register
TEST_TYPE:    ANDI   D0, 0XF0, D1           ;Test the op code
              CMPI   D1, 0Xdata0            ;Is this a packet?
              BZ     CONT_HDR               ;Continue processing if so
              MOVEI        0X300000000, D3  ;Else set the cycle start flag
              MOVE   DATA_3, D0             ;And get the next quadlet
              BRA    TEST_TYPE              ;Then test this one also
CONT_HDR:     ANDI   D0, 0X3F00, D1         ;get the ch number
              ANDI   D0, 0XFFFFC00F, D3     ;Clear tcode and ch fields
              ORI    D0, 0X'isoch'0, D3     ;restore tcode
              CMPI   D1, 0X700              ;is this channel 7?
              BZ     CH_7                   ;Branch to handle if so
              CMPI   D1, 0X900              ;is this channel9?
              BNZ    DISCARD                ;discard if not
              ORI    D3, 0X200, BUS_OUT     ;Else this is ch 9
              BRA    CONT_DATA              ;Then continue with data field
CH_7:         ORI    D3, 0X100, BUS OUT     ;map to channel 1
CONT_DATA:    BSR    GET_COUNT              ;Get quadlet count
CONT_OUT:     BZ     CS_output              ;continue if not
              MOVE   DATA_3, BUS_OUT        ;send something out
              SUBI   D3, 4, D3              ;decrement quadlet counter
              BRA    CONT_OUT               ;and continue outputting
DISCARD:      BSR    GET_COUNT              ;Get quadlet count
CONT_DIS:     BZ     CS_output              ;Continue if not
              MOVE   DATA_3 DO              ;Else, get a quadlet
              SUBI   D3, 4, D3              ;Decrement quadlet counter
              BRA    CONT_DIS               ;Continue to discard
GET_COUNT:    SHIFTI D3, 16, D3             ;Get the data length
              ADDI   D3, 3, D3              ;Wrap it up
              ANDI   D3, FFFC, D3           ;And mask
              RET                           ;then return
```

Formats for carrying digital consumer audio and video data over an IEEE 1394 format bus via an isochronous channel contain absolute time stamps which are inserted by the sender and used at the receiver to recreate the timing information necessary to decode the stream of data. Similarly, non-consumer audio and video storage devices, such as a hard disk drive, will also modify this embedded time stamp information such that when the data is played back at a later time, a consumer device receiving the data will function properly.

The isochronous data pipe **20** is first initialized by an application before it can transfer isochronous data. Once initialized, the application uses the control register to change the operational state of the isochronous data pipe **20**. The current operational state is completely defined by the value in the current channel mask register and the information contained in the low order byte of the status register.

In order to change the state of the isochronous data pipe **20**, the application programs a new channel mask value into the pending channel mask register and a new operational state into the control register. In the same register access to the control register, the application also sets the go bit and programs an event into the event field. When the programmed event occurs, the isochronous data pipe **20** transfers the value in the pending channel mask register to the current channel mask register. The isochronous data pipe **20** also transfers the information in the low order byte of the control register into the low order byte of the status register.

In order to stop the operation of the isochronous data pipe **20**, the application programs a value of one into the pending channel mask register and a logical low voltage level into the transmit enable bit in the control register. As with any state change, the application also sets the go bit and programs an event into the event field of the control register. When the event occurs the value in the current channel mask register becomes zero and the transmit enable bit in the status register is pulled to a logical low voltage level, thereby stopping the operation of the isochronous data pipe **20**. When the operation of the isochronous data pipe **20** is stopped, the active bit in the status register is also pulled to a logical low voltage level.

In order to activate the isochronous data pipe **20**, the application first loads a control store program and programs the proper control store offsets into the control store address register file. The pending channel mask register is then programmed with a bit mask of the channels which the isochronous data pipe **20** is to receive. If the isochronous data pipe **20** is not receiving data, the value in the pending channel mask register is programmed to zero. The go bit in the control register is set to a logical high voltage level to indicate a state change. The transmit enable bit in the control register is set to a logical high voltage level if the isochronous data pipe **20** is transmitting isochronous data. The stop on error bit in the control register is set to a logical high voltage level if the isochronous data pipe **20** is to stop operation on any error. If the isochronous data pipe **20** is transmitting isochronous data, the output enable bits in the control register which correspond to the DMA channels involved in transmitting isochronous data are set to a logical high voltage level. The event field in the control register is programmed to an event on which the isochronous data pipe **20** is to change state.

The asynchronous data pipe **26**, as stated above, automatically generates transactions necessary to complete asynchronous data transfer operations for an application over the bus structure **58**. In an alternate embodiment of the isochronous data pipe **20** of the present invention, the isochronous data pipe **20** can be programmed to transfer and receive both isochronous and asynchronous data. Accordingly, in this

embodiment, the asynchronous data pipe 26 and the corresponding FIFO 32 are not necessary. Furthermore, because the isochronous data pipe 20 is the only logical block within the link circuit, the multiplexer 40 and demultiplexer 42 are also not necessary.

As described above, the isochronous data pipe 20 of the preferred embodiment is programmed to execute a control store program and perform a series of operations on a stream of isochronous data. In this alternate embodiment, the isochronous data pipe 20 also can be programmed to send and receive asynchronous data. In this embodiment, the isochronous data pipe 20 appears as a virtual asynchronous data pipe and is programmed to generate the transactions necessary to complete asynchronous data transfer operations, as well as generate the appropriate headers when sending data and strip headers from received data, as described in U.S. patent application Ser. No. 08/612,321, filed on the same date as the present application and entitled "Asynchronous Data Pipe For Automatically Managing Asynchronous Data Transfers Between An Application And A Bus Structure."

In this alternate embodiment, the isochronous data pipe 20 will send or receive both isochronous and asynchronous data. The isochronous data pipe 20 is programmed by an application to execute an appropriate program for manipulating either an isochronous or asynchronous stream of data, as necessary. When receiving or transmitting asynchronous data the isochronous data pipe 20 is programmed to automatically generate the read or write transactions necessary to complete the data transfer operation over the appropriate range of addresses. The isochronous data pipe will appropriately automatically increment the value in the destination offset address field for each transaction according to the length of each data packet, unless an incrementing feature has been disabled, signalling that the transactions are to take place at a single address.

The present invention has been described in terms of specific embodiments incorporating details to facilitate the understanding of the principles of construction and operation of the invention. Such reference herein to specific embodiments and details thereof is not intended to limit the scope of the claims appended hereto. It will be apparent to those skilled in the art that modifications may be made in the embodiment chosen for illustration without departing from the spirit and scope of the invention.

We claim:

1. A method of controlling streams of data between an application and a bus structure comprising:
   a. receiving a stream of data from a source selected from the application and the bus structure;
   b. determining, if the received stream of data is isochronous, a channel number and an address corresponding to the channel number;
   c. determining, if the received stream of data is asynchronous, an address corresponding thereto;
   d. obtaining a series of at least one operation codes from the address determined; and
   e. generating an output stream of data by converting the received stream of data into the output stream of data by executing the series of operation codes.

2. The method as claimed in claim 1 wherein the bus structure is an IEEE 1394 standard bus structure.

3. An apparatus for controlling bidirectional streams of data between an application and a bus structure comprising:
   a. means for receiving a stream of data from a source selected from the application and the bus structure;
   b. means for storing a series of at least one operation codes corresponding to the received stream of data when the received stream is asynchronous and for storing at least one series of at least one operation codes corresponding to a channel number of the received stream when the received stream is isochronous; and
   c. means for generating an output stream of data by converting the received stream of data into the output stream of data by executing the series of operation codes corresponding to the received stream of data.

4. The apparatus as claimed in claim 3 wherein if the received stream of data is received from the application, the output stream of data is provided to the bus structure, and if the received stream of data is received from the bus structure, the output stream of data is provided to the application.

5. The apparatus as claimed in claim 3 wherein the bus structure is an IEEE 1394 standard bus structure.

6. An apparatus for controlling and managing data transfer operations between at least one application and a bus structure comprising:
   a. an isochronous data processing apparatus configured for coupling between one of the applications and the bus structure, including:
      i. means for receiving a stream of data from a source selected from one of the applications and the bus structure and forming a received stream of data;
      ii. means for obtaining a series of at least one operation codes regarding the received stream of data; and
      iii. means for converting the received stream of data into the output stream of data by executing the series of operation codes;
   b. a physical bus interface configured for coupling to the bus structure for placing data on the bus structure and obtaining data from the bus structure;
   c. a memory coupled to the means for obtaining and configured for coupling to the application for storing the series of operation codes;
   d. an asynchronous data pipe for controlling asynchronous data transfer operations over the bus structure including:
      i. means for receiving instructions configured for coupling to the application for receiving instructions regarding an asynchronous data transfer operation; and
      ii. means for generating transactions necessary to complete the asynchronous data transfer operation between the application and a node coupled to the bus structure; and
   e. a multiplexing circuit coupled to the isochronous data processing apparatus, the asynchronous data pipe and the physical bus interface for transmitting data packets from the isochronous data processing apparatus and the asynchronous data pipe to the bus structure.

7. The apparatus as claimed in claim 6 further comprising a demultiplexing circuit coupled to the isochronous data processing apparatus, the asynchronous data pipe and the physical bus interface for routing data packets obtained from the bus structure to an appropriate one of the isochronous data processing apparatus and asynchronous data pipe.

8. The apparatus as claimed in claim 6 wherein the bus structure is an IEEE 1394 standard bus structure.

9. The apparatus as claimed in claim 6 wherein the isochronous data processing apparatus will execute series of operation codes regarding both isochronous and asynchronous received streams of data.

10. The apparatus as claimed in claim 9 wherein the series of operation codes is stored at a memory address corre-

sponding to a channel number on which the received stream of data is transmitted if the received stream of data is an isochronous stream of data and at a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

11. The apparatus as claimed in claim 10 wherein the bus structure is an IEEE 1394 standard bus structure.

12. An isochronous data processing apparatus configured to couple between an application and an IEEE 1394 standard bus structure to manage isochronous data transfer operations over the bus structure comprising:

a. a receiving circuit configured to receive a stream of data from a source selected from the application and the IEEE 1394 standard bus structure and to form a received stream of data;

b. a control store memory, wherein the application stores a series of at least one operation codes to manipulate an isochronous stream of data wherein the series of operation codes is stored corresponding to a channel number; and

c. a converting circuit coupled to the control store memory and to the receiving circuit to obtain the series of operation codes and convert the received stream of data into the output stream of data by executing the series of operation codes.

13. A method of controlling streams of data between an application and a bus structure comprising:

a. receiving a stream of data from a source selected from the application and the bus structure thereby forming a received stream of data, wherein the received stream of data is one of an isochronous stream of data and an asynchronous stream of data;

b. obtaining a series of at least one operation codes regarding the received stream of data;

c. generating an output stream of data by manipulating the received stream of data by executing the series of operation codes;

d. transmitting the output stream of data to at least one appropriate destination selected from the application and a node on the bus structure;

e. determining if the received stream of data is an isochronous stream of data or an asynchronous stream of data; and

f. determining a channel number for the received stream of data, if the received stream of data is an isochronous stream of data;

wherein the series of operation codes is obtained from a memory address corresponding to the channel number if the received stream of data is an isochronous stream of data and from a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

14. The method as claimed in claim 13 wherein the bus structure is an IEEE 1394 standard bus structure.

15. An apparatus to control and manage data transfer operations between at least one application and a bus structure comprising:

a. an isochronous data processing apparatus configured to couple between at least one application and the bus structure, including:

i. an interface circuit configured to receive a stream of data from a source selected from the application and the bus structure to form a received stream of data and to obtain a series of at least one operation codes regarding the received stream of data; and

ii. a converting circuit configured to convert the received stream of data into the output stream of data by executing the series of operation codes;

b. a physical bus interface configured to couple to the bus structure to place the output stream of data on the bus structure if the source is the application and obtain the received stream of data from the bus structure if the source is the bus structure; and

c. a memory coupled to the interface circuit to store the series of operation codes, wherein the series of operation codes is stored at a memory address corresponding to a channel number on which the received stream of data is transmitted.

16. The apparatus as claimed in claim 15 wherein the bus structure is an IEEE 1394 standard bus structure.

17. An apparatus to control bidirectional streams of data between an application and a bus structure comprising:

a. a receiving circuit configured to receive a stream of data from a source selected from the application and the bus structure and form a received stream of data, to obtain a series of at least one operation codes regarding the received stream of data and to determine if the received stream of data is an isochronous stream of data or an asynchronous stream of data;

b. a converting circuit coupled to the receiving circuit and configured to convert the received stream of data into an output stream of data by executing the series of operation codes; and

c. a memory coupled to the receiving circuit and the converting circuit to store the series of operation codes at a memory address corresponding to a channel number on which the received stream of data is transmitted if the received stream of data is an isochronous stream of data and at a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

18. A method of controlling a stream of data between an application and a bus structure comprising:

a. receiving a stream of data from the application thereby forming a received stream of data, wherein the received stream of data is one of an isochronous stream of data and an asynchronous stream of data;

b. obtaining a series of at least one operation codes regarding the received stream of data;

c. generating an output stream of data by converting the received stream of data into the output stream of data by executing the series of operation codes;

d. determining if the received stream of data is an isochronous stream of data or an asynchronous stream of data; and

e. determining a channel number for the received stream of data, if the received stream of data is an isochronous stream of data;

wherein the series of operation codes is obtained from a memory address corresponding to the channel number if the received stream of data is an isochronous stream of data and from a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

19. A method of controlling a stream of data between a bus structure and an application comprising:

a. receiving a stream of data from the bus structure thereby forming a received stream of data, wherein the received stream of data is one of an isochronous stream of data and an asynchronous stream of data;

b. obtaining a series of at least one operation codes regarding the received stream of data;

c. generating an output stream of data by converting the received stream of data into the output stream of data by executing the series of operation codes;

d. determining if the received stream of data is an isochronous stream of data or an asynchronous stream of data; and

e. determining a channel number for the received stream of data, if the received stream of data is an isochronous stream of data;

wherein the series of operation codes is obtained from a memory address corresponding to the channel number if the received stream of data is an isochronous stream of data and from a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

20. An apparatus for controlling a stream of data between an application and a bus structure comprising:

a. means for receiving a stream of data from the application and forming a received stream of data;

b. means for obtaining a series of at least one operation codes regarding the received stream of data;

c. means for converting the received stream of data into an output stream of data by executing the series of operation codes;

d. means for determining coupled to the means for receiving for determining if the received stream of data is an isochronous stream of data or an asynchronous stream of data; and

e. means for storing coupled to the means for obtaining for storing the series of operation codes at a memory address corresponding to a channel number on which the received stream of data is transmitted if the received stream of data is an isochronous stream of data and at a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

21. An apparatus for controlling a stream of data between a bus structure and an application comprising:

a. means for receiving a stream of data from the bus structure and forming a received stream of data;

b. means for obtaining a series of at least one operation codes regarding the received stream of data;

c. means for converting the received stream of data into the output stream of data by executing the series of operation codes;

d. means for determining coupled to the means for receiving for determining if the received stream of data is an isochronous stream of data or an asynchronous stream of data; and

e. means for storing coupled to the means for obtaining for storing the series of operation codes at a memory address corresponding to a channel number on which the received stream of data is transmitted if the received stream of data is an isochronous stream of data and at a memory address corresponding to asynchronous data if the received stream of data is an asynchronous stream of data.

22. The apparatus as claimed in claim 6 wherein the isochronous data processing apparatus is an embedded stream processor.

23. The isochronous data processing apparatus as claimed in claim 12 wherein the isochronous data processing apparatus is an embedded stream processor.

24. An apparatus for controlling streams of data between an application and a bus structure comprising:

a. a receiving circuit configured to receive a stream of data from a source selected from the application and the bus structure;

b. an addressing circuit configured to obtain, if the received stream of data is asynchronous, a series of at least one operation code corresponding to the received stream, and configured to obtain, if the received stream is isochronous, a series of at least one operation code corresponding to a channel number of the received stream; and

c. a converting circuit configured to execute the obtained series of operation codes thereby converting the received stream into an output stream.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,233,637 B1
DATED : May 15, 2001
INVENTOR(S) : Smyers et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [56], add --

5,369,773	12/97	Wu et. al.	370/380
5,659,780	08/97	Wu	395/800.19
5,704,052	11/94	Hammerstrom et. al.	395/800

Delete "Anderson et al." and insert -- Andersen et al. --.

Signed and Sealed this

Thirteenth Day of November, 2001

Attest:

Nicholas P. Godici

Attesting Officer

NICHOLAS P. GODICI
Acting Director of the United States Patent and Trademark Office